

Deployment options for OSGi applications in the cloud/edge

Deployment options for OSGi applications in the cloud/edge

Speaker



Dirk Fauth

*Research Engineer
Eclipse Committer*

ETAS GmbH
Borsigstraße 24
70469 Stuttgart

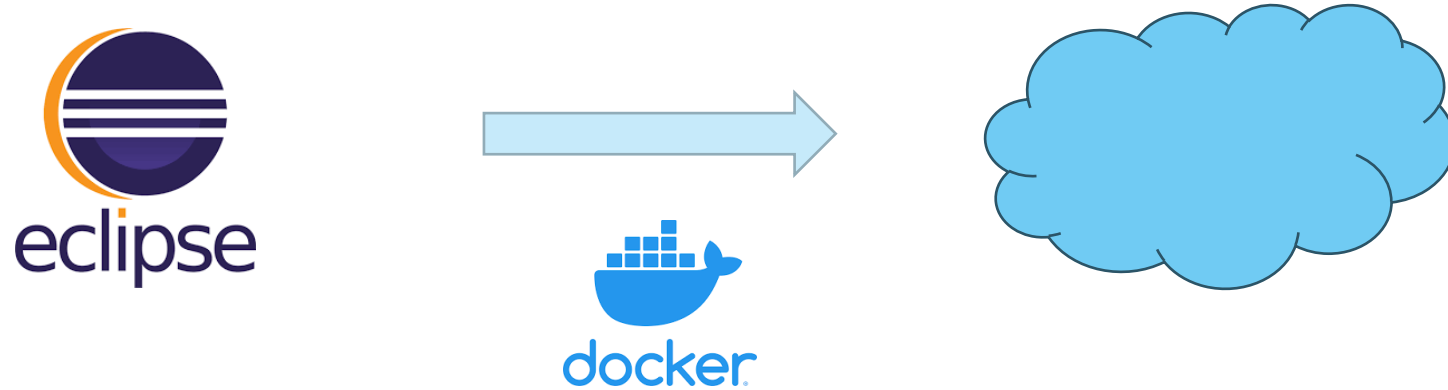
dirk.fauth@etas.com
www.etas.com

blog.vogella.com/author/fipro/
Twitter: [fipro78](#)

Deployment options for OSGi applications in the cloud/edge

Motivation

Shift existing Eclipse application (partly) to the cloud



„Startup of a Java application to slow for cloud applications!“

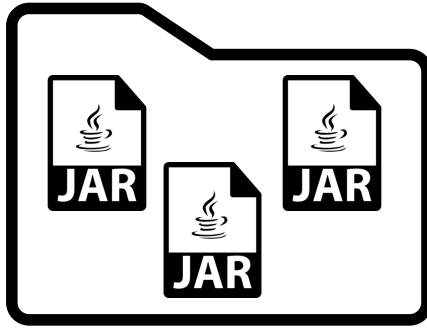
„Size of a container for a Java application to big!“

- 1. Deployment Variants**
- 2. Container**
- 3. Benchmark**
- 4. Conclusion**

Deployment Variants

Deployment Variants

General



Multiple JARs in a folder



Executable JAR



Custom JRE (jlink)

GraalVM™

Native Executable

Deployment Variants

Multiple JARs in a folder

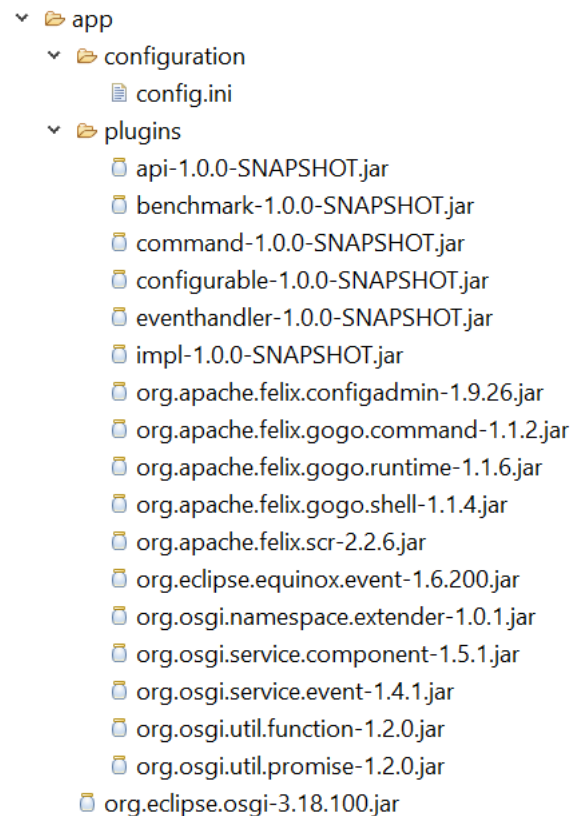
- Multiple JAR files (OSGi bundles) inside a folder
- Additional configuration file
- Launcher

```
org.eclipse.osgi
```

```
:org.eclipse.core.runtime.adaptor.EclipseStarter
```

```
java -jar org.eclipse.osgi-3.18.100.jar
```

- Build
 - maven-dependency-plugin
 - maven-resources-plugin



<https://www.eclipse.org/equinox/documents/quickstart-framework.php>

Executable JAR

- Executable JAR that includes each required bundle as embedded JAR file
- Configuration also included in the executable JAR
- Launcher

```
aQute.launcher.pre.EmbeddedLauncher
```

```
java -jar equinox-app.jar
```

- Build
 - bnd-maven-plugin
 - bnd-export-maven-plugin



<https://bnd.bndtools.org/>

<https://bndtools.org/>

<https://github.com/bndtools/bnd/tree/master/maven-plugins>

Custom JRE via jlink

- Create a custom JRE with `jlink` command of the JDK
 - *assemble and optimize a set of **modules** and their dependencies into a custom runtime image*

<https://docs.oracle.com/en/java/javase/17/docs/specs/man/jlink.html>

- Folder layout like JRE
- Launcher: `java` command

```
java [options] -m <module>[/<mainclass>]
```

```
/app/jre $ ls -l
total 20
drwxr-xr-x  2 appuser appuser 4096 Oct 14 08:37 bin
drwxr-xr-x  4 appuser appuser 4096 Oct 14 08:37 conf
drwxr-xr-x  9 appuser appuser 4096 Oct 14 08:37 legal
drwxr-xr-x  4 appuser appuser 4096 Oct 14 08:37 lib
-rw-r--r--  1 appuser appuser  140 Oct 14 08:37 release
/app/jre $
```

- Issue with OSGi and jlink
Most available OSGi bundles do not contain a `module-info.class`
→ **automatic module cannot be used with jlink**

JPMS

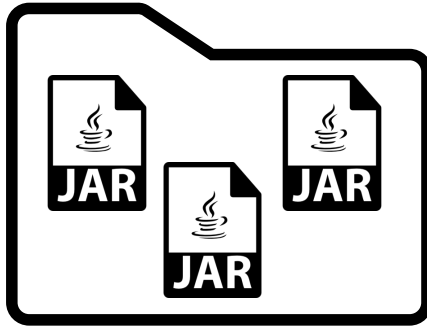
Native Executable with GraalVM

- *Native Image is a technology to compile Java code ahead-of-time to a binary – a native executable. A native executable includes only the code required at run time, that is the application classes, standard-library classes, the language runtime, and statically-linked native code from the JDK.*
- Can be created using the GraalVM `native-image` tool
 - From a **Class**, a **JAR (classpath)** or a **Module (modulepath)**
- “Closed world assumption”
 - all the bytecode in your application that can be called at run time must be known at build time
- Issue with OSGi and `native-image`
Dynamic classloading per bundle managed by OSGi Framework (Module Layer)
`java.lang.NullPointerException: A null service reference is not allowed.`

<https://www.graalvm.org/reference-manual/native-image/>

Deployment Variants

OSGi



Multiple JARs in a folder



Executable JAR



Custom JRE (jlink)



Native Executable

Deployment Variants

Custom JRE via jlink - OSGi

– Add `module-info.class`

– ModiTect

<https://github.com/moditect/moditect>

→ Intrusive change that adds an artifact to an existing published JAR

OSS license compatibility?

Checksum?

→ Requires knowledge on internals for generation

Maintenance?

– **Bndtools JPMS Support**

<https://bnd.bndtools.org/chapters/330-jpms.html>



Enable creation of `module-info.class` for each bundle, e.g. via `bnd-maven-plugin`

```
<plugin>
  <groupId>biz.aQute.bnd</groupId>
  <artifactId>bnd-maven-plugin</artifactId>
  <configuration>
    <bnd>
      <![CDATA[
Bundle-SymbolicName: ${project.groupId}.${project.artifactId}
-sources: true
-contract: *
-jpms-module-info:org.fipro.service.command;modules='org.apache.felix.configadmin'
-jpms-module-info-options: org.osgi.service.cm;ignore="true"
]]>
    </bnd>
  </configuration>
</plugin>
```

Enable creation of `module-info.class` for **executable jar** via `.bndrun` file

```
-jpms-module-info: \  
    ${project.groupId}.equinox.${project.artifactId};\  
    version=${project.version};\  
    ee=JavaSE-${java.specification.version}  
-jpms-module-info-options: jdk.unsupported;static=false
```

This makes the executable jar itself a module!

Deployment Variants

Custom JRE via jlink with Bndtools JPMS support

Build

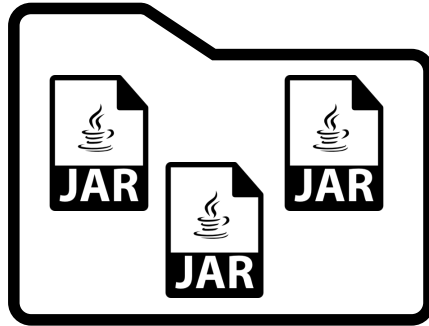
```
$JAVA_HOME/bin/jlink \  
  --add-modules org.fipro.service.equinox.app \  
  --module-path equinox-app.jar \  
  --no-header-files \  
  --no-man-pages \  
  --output /app/jre
```

Launch

```
/app/jre/bin/java \  
  -m org.fipro.service.equinox.app/aQute.launcher.pre.EmbeddedLauncher
```

Deployment Variants

OSGi



+



Multiple JARs in a folder



+



Executable JAR



Custom JRE (jlink)



Native Executable

OSGi Connect

– *OSGi Connect allows for bundles to exist and be installed into the OSGi Framework from the flat class path, the module path (Java Platform Module System), a jlink image, or a native image.*

→ Allows to start an OSGi application without the full OSGi Module Layer

OSGi Core R8 – Connect Specification

<https://docs.osgi.org/specification/osgi.core/8.0.0/framework.connect.html>

Apache Felix Atomos

<https://github.com/apache/felix-atomos>

Ubiquitous OSGi - Android, Graal Substrate, Java Modules, Flat Class Path

<https://www.youtube.com/watch?v=KxmtzjHBumU>

OSGi R8, Felix 7, Atomos and the future of OSGi@Eclipse

<https://www.youtube.com/watch?v=oitFMbzt5s>

GraalVM Native Image with OSGi Connect

– Preparation

1. Add/use Atomos to be able to start the OSGi application from the flat classpath
2. Generate reachability metadata via tracing agent (reflection, resources, ...)
3. Update generated metadata

– Build

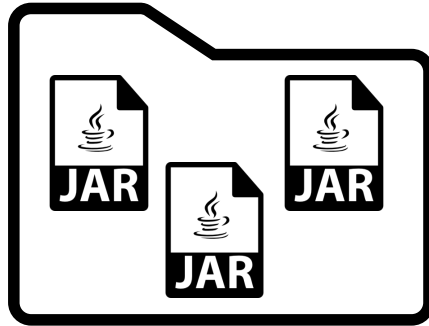
- Via GraalVM build plugins (Maven/Gradle)
- Docker multi-stage build using GraalVM container images

– Notes/Remarks

- `native-image` build only worked with **flat classpath** and **listing all jars explicitly**
- Build result is platform-dependent
- `atomos_lib` folder or index file needed for Atomos to discover bundles and load bundle entries
- Still not everything is working as expected (e.g. `scr:list` produces an empty output)

Deployment Variants

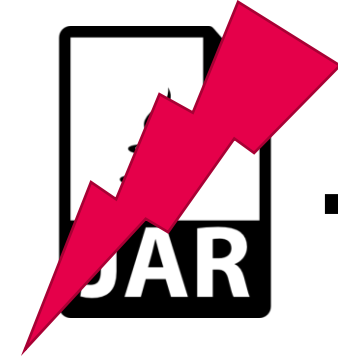
OSGi Connect / Apache Felix Atomos



+



Multiple JARs in a folder



+



Executable JAR



Custom JRE (jlink)

GraalVM™

Native Executable

Deployment (plain OSGi)

Multiple JARs in folder

Executable JAR

Custom JRE (jlink)

GraalVM Native Image

Deployment (OSGi Connect)

Multiple JARs in folder

Executable JAR

Custom JRE (jlink)

GraalVM Native Image

Container

“Size matters” – Find the right base image

Alpine vs. Debian vs. Ubuntu

Image	Size
alpine:3	7.33 MB
debian:bullseye-slim	80.55 MB
ubuntu:jammy	77.82 MB

Eclipse Temurin vs. IBM Semeru JDK vs. JRE

Image	Size (11)	Size (17)	Size (21)
eclipse-temurin:xx-jdk-jammy	~ 392 MB	~ 407 MB	~ 435 MB
eclipse-temurin:xx-jdk-alpine	~ 290 MB	~ 304 MB	~ 332 MB
eclipse-temurin:xx-jre-jammy	~ 254 MB	~ 269 MB	~ 292 MB
eclipse-temurin:xx-jre-alpine	~ 154 MB	~ 167 MB	~ 190 MB
ibm-semeru-runtimes:open-xx-jdk-jammy	~ 473 MB	~ 482 MB	
ibm-semeru-runtimes:open-xx-jre-jammy	~ 274 MB	~ 275 MB	

** images pulled on 2023/12/10*

Interlude: Distroless

- *"Distroless" images contain only your application and its runtime dependencies. They do not contain package managers, shells or any other programs you would expect to find in a standard Linux distribution.*

Image		Size
gcr.io/distroless/static-debian12	minimal Linux for "mostly-statically compiled" languages that do not require libc	1.98 MB
gcr.io/distroless/base-debian12	minimal Linux, glibc-based system	20.68 MB
gcr.io/distroless/java17-debian12	base image plus OpenJDK 17 and its dependencies	225.66 MB

- Distroless Java image is based on Debian and glibc, therefore bigger than an Alpine Temurin image
- Can be interesting in production for security reasons, but not for size

<https://github.com/GoogleContainerTools/distroless>

Java Best Practices

- Install only what you need
 - Use JRE instead of JDK
 - Use multi-stage builds (e.g. to create JRE or Native Image)
- Don't run Java apps as root
- Properly shutdown and handle events to terminate a Java application
- Take care of “container-awareness”

<https://snyk.io/blog/best-practices-to-build-java-containers-with-docker/>

<https://developers.redhat.com/articles/2022/04/19/java-17-whats-new-openjdk-container-awareness#>

<https://blog.openj9.org/2021/06/15/innovations-for-java-running-in-containers/>

Building Docker Images

- Use dedicated Docker files instead of generation tools
- Integrate image creation as part of the build via **fabric8io/docker-maven-plugin**
Maven/Gradle first

<https://github.com/fabric8io/docker-maven-plugin>
<http://dmp.fabric8.io/>

```
<plugin>
  <groupId>io.fabric8</groupId>
  <artifactId>docker-maven-plugin</artifactId>
  <extensions>true</extensions>
  <configuration>
    <images> ... </images>
  </configuration>
  <executions> ... </executions>
</plugin>
```

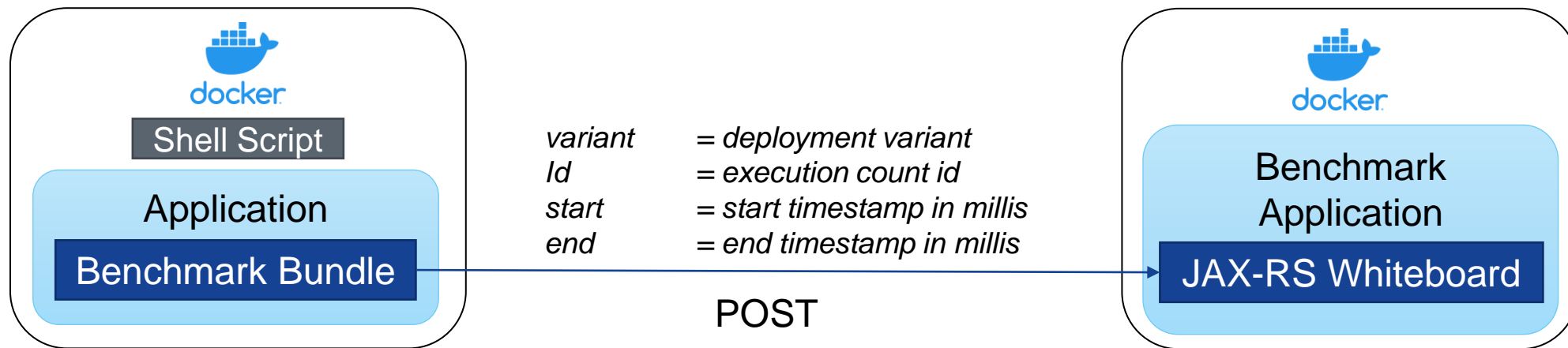
- Use multi-stage build to checkout sources and build in one container, then create new production container with build result only
Docker first

Deployment Variant – Base Image – Image Size

Deployment (plain OSGi)	Base Image	Size
Multiple JARs in folder	eclipse-temurin:17-jre-alpine	~ 169 MB
Executable JAR	eclipse-temurin:17-jre-alpine	~ 170 MB
Custom JRE (jlink)	alpine:3	~ 78 MB
Custom JRE (jlink/compressed)	alpine:3	~ 56 MB

Deployment (OSGi Connect)	Base Image	Size
Multiple JARs in folder	eclipse-temurin:17-jre-alpine	~ 169 MB
Custom JRE (jlink)	alpine:3	~ 78 MB
Custom JRE (jlink/compressed)	alpine:3	~ 56 MB
GraalVM Native Image	scratch	~ 38 MB
	alpine:3	~ 43 MB

Benchmark



Benchmark Bundle / Immediate Component

- Get start timestamp from system property
- Get current timestamp
- Send POST request via `java.net.http.HttpClient`
- Shutdown

Shell script

- Execute application multiple times in for-loop (clean/cache)
- Pass start timestamp as system property

Benchmark Images Java 17

Deployment Variant – Base Image – Image Size – Benchmark Image Size

Deployment (plain OSGi)	Base Image	Size	Size Benchmark
Multiple JARs in folder	eclipse-temurin:17-jre-alpine	~ 169 MB	~ 171 MB
Executable JAR	eclipse-temurin:17-jre-alpine	~ 170 MB	~ 172 MB
Custom JRE (jlink)	alpine:3	~ 78 MB	~ 81 MB
Custom JRE (jlink/compressed)	alpine:3	~ 56 MB	~ 58 MB
Deployment (OSGi Connect)	Base Image	Size	Size Benchmark
Multiple JARs in folder	eclipse-temurin:17-jre-alpine	~ 169 MB	~ 171 MB
Custom JRE (jlink)	alpine:3	~ 78 MB	~ 81 MB
Custom JRE (jlink/compressed)	alpine:3	~ 56 MB	~ 58 MB
Oracle GraalVM 17 Native Image	scratch alpine:3	~ 47 MB (~ 52 MB)	(~ 58 MB) ~ 65 MB
GraalVM CE 17 Native Image	scratch alpine:3	~ 41 MB (~ 46 MB)	(~ 50 MB) ~ 58 MB

- + coreutils
- + nanosecond support
- + benchmark bundle
- + java.net.http module
- + shell script support

Deployment Variant – Base Image – Benchmark Image Size

Deployment (plain OSGi)	Base Image	Size (11)	Size (17)	Size (21)
Multiple JARs in folder	eclipse-temurin:xx-jre-alpine	~ 159 MB	~ 171 MB	~ 194 MB
Executable JAR	eclipse-temurin:xx-jre-alpine	~ 160 MB	~ 172 MB	~ 195 MB
Custom JRE (jlink)	alpine:3	~ 79 MB	~ 81 MB	~ 87 MB
Custom JRE (jlink/compressed)	alpine:3	~ 57 MB	~ 58 MB	~ 62 MB
Deployment (OSGi Connect)	Base Image	Size	Size (17)	Size (21)
Multiple JARs in folder	eclipse-temurin:21-jre-alpine	~ 159 MB	~ 171 MB	~ 194 MB
Custom JRE (jlink)	alpine:3	~ 79 MB	~ 81 MB	~ 87 MB
Custom JRE (jlink/compressed)	alpine:3	~ 57 MB	~ 58 MB	~ 62 MB
Oracle GraalVM xx Native Image	scratch alpine:3		(~ 58 MB) ~ 65 MB	(~ 57 MB) ~ 64 MB
GraalVM CE xx Native Image	scratch alpine:3		(~ 50 MB) ~ 58 MB	(~ 53 MB) ~ 60 MB

Benchmark Results

Java 11

Java 17

Java 21

Deployment (plain OSGi)	Startup Clean	Startup Cache	Startup Clean	Startup Cache	Startup Clean	Startup Cache
Multiple JARs in folder	~ 1136 ms	~ 1070 ms	~ 984 ms	~ 1019 ms	~ 920 ms	~ 1023 ms
Executable JAR	~ 1208 ms	~ 1187 ms	~ 1086 ms	~ 1149 ms	~ 1073 ms	~ 1112 ms
Custom JRE (jlink)	~ 1387 ms	~ 1433 ms	~ 1425 ms	~ 1409 ms	~ 1315 ms	~ 1355 ms
Custom JRE (jlink/compressed)	~ 1589 ms	~ 1556 ms	~ 1511 ms	~ 1489 ms	~ 1455 ms	~ 1464 ms
Deployment (OSGi Connect)	Startup Clean	Startup Cache	Startup Clean	Startup Cache	Startup Clean	Startup Cache
Multiple JARs in folder						
classpath	~ 1611 ms	~ 1098 ms	~ 1479 ms	~ 960 ms	~ 1275 ms	~ 988 ms
modulepath	~ 1494 ms	~ 1163 ms	~ 1450 ms	~ 1072 ms	~ 1204 ms	~ 1020 ms
Custom JRE (jlink)	~ 1411 ms	~ 1436 ms	~ 1394 ms	~ 1350 ms	~ 1293 ms	~ 1308 ms
Custom JRE (jlink/compressed)	~ 1598 ms	~ 1556 ms	~ 1526 ms	~ 1528 ms	~ 1441 ms	~ 1442 ms
Oracle GraalVM Native Image			-	-	-	-
			~ 52 ms	~ 45 ms	~ 49 ms	~ 29 ms
GraalVM CE Native Image			-	-	-	-
			~ 58 ms	~ 63 ms	~ 62 ms	~ 38 ms

Conclusion

- All Java deployment variants possible for OSGi applications via
 - Bndtools JPMS support
 - OSGi Connect (Felix Atomos)
- Different deployment variants have different startup & runtime behaviors
Consider JIT vs. AOT compilation
- Newer Java versions have bigger container sizes
- Make decision about variant dependent on the use case,
e.g. short running executables in container vs. long running application servers
- Further optimizations possible by configuring the Java runtime,
e.g. Container-awareness, Garbage Collection, *Checkpoint & Restore*, etc.

Appendix

OpenJ9 – CRIU – CRaC

Image Size

- No Alpine version
- RedHat Universal Base Image (UBI) is recommended

Image icr.io/appcafe/ibm-semeru-runtimes	Size
:open-17-jre-jammy	274.80 MB
:open-17-jdk-jammy	482.15 MB
:open-17-jre-ubi9-minimal	291.36 MB
:open-17-jdk-ubi9-minimal	498.72 MB

Startup Performance

-Xquickstart

-Xshareclasses

Deployment (plain OSGi)	Base Image	Size	Startup Clean	Startup Cache	Startup Clean	Startup Cache	Size	Startup Clean	Startup Cache
Multiple JARs in folder	open-17-jre-jammy	~ 278 MB	~ 1101 ms	~ 1017 ms	~ 1052 ms	~ 1022 ms	~ 299 MB	~ 721 ms	~ 641 ms
Executable JAR	open-17-jre-jammy	~ 279 MB	~ 1288 ms	~ 1200 ms	~ 1222 ms	~ 1198 ms	~ 301 MB	~ 1004 ms	~ 1010 ms
Custom JRE (jlink)	debian:bullseye-slim	~ 166 MB	~ 2591 ms	~ 2604 ms	~ 1656 ms	~ 1645 ms	~ 189 MB	~ 1083 ms	~ 1098 ms
Custom JRE (jlink/compressed)	debian:bullseye-slim	~ 143 MB	~ 2748 ms	~ 2777 ms	~ 1777 ms	~ 1794 ms	~ 166 MB	~ 1177 ms	~ 1200 ms

Deployment (OSGi Connect)	Base Image	Size	Startup Clean	Startup Cache	Startup Clean	Startup Cache	Size	Startup Clean	Startup Cache
Multiple JARs in folder classpath modulepath	open-17-jre-jammy	~ 278 MB	~ 1041 ms ~ 1126 ms	~1039 ms ~ 1124 ms	~ 1016 ms ~ 1097 ms	~1021 ms ~ 1108 ms	~302 MB	~ 734 ms ~ 831 ms	~750 ms ~ 801 ms
Custom JRE (jlink)	debian:bullseye-slim	~ 166 MB	~2492 ms	~2562 ms	~1603 ms	~1630 ms	~190 MB	~1110 ms	~1180 ms
Custom JRE (jlink/compressed)	debian:bullseye-slim	~ 143 MB	~ 3309 ms	~2714 ms	~ 1714 ms	~1725 ms	~167 MB	~ 1170 ms	~1224 ms

“AOT like startup performance with JIT runtime performance and behaviour”

- **CRaC (Coordinated Restore at Checkpoint)**
 - OpenJDK CRaC JDK
 - Azul Zulu JDK with CRaC support
 - Enable via `-XX:CRaCCheckpointTo=<path>`
 - Create a checkpoint via `jdk.crac.Core` API
 - Create a checkpoint via `jcmd <PID> JDK.checkpoint` from separate shell process
- **OpenJ9 CRIU support / Open Liberty InstantOn**
 - IBM Semeru (OpenJDK/OpenJ9)
 - Enable via `-XX:+EnableCRIUSupport`
 - Creating a checkpoint via `org.eclipse.openj9.criu.CRIUSupport` API

Current issues


– CRaC (Coordinated Restore at Checkpoint)

- Only Linux, currently no Alpine (musl) variant available
- Building a custom JRE (jlink) currently not easily possible (need to manually copy criu to the system)
- Containers with CRaC support not yet available out-of-the-box

- 
- **Checkpoint can not be created with open file handles**
 - **Equinox has open file handles to the bundle jar files**

– OpenJ9 CRIU support / Open Liberty InstantOn

- Only Linux, no Alpine
- Building a custom JRE (jlink)?
- Creating a checkpoint only possible via API

- 
- **Checkpoint creation fails with exceptions using Docker on Windows**

– April 2023

Azul Zulu builds with CRaC support generally available for Java 17 on Linux

– June 2023

Open Liberty InstantOn out of beta with release 23.0.0.6

– June 2023

Oracle announced **Oracle GraalVM** distributed under **GraalVM Free License** with release 23.0

→ Oracle GraalVM Enterprise now available for free

Be careful: *“You do not charge Your licensees any fees”*

https://github.com/fipro78/osgi_deployment_options

Thank you

Dirk Fauth

ETAS/ENA

dirk.fauth@etas.com